On Software Implementation of Arithmetic Operations on Prime Fields using AVX2

Armando Faz-Hernández, Julio López.

[1]Institute of Computing, University of Campinas.
{armfazh,jlopez}@ic.unicamp.br

Abstract. *AVX2 is the newest instruction set on Intel Haswell processor that provides simultaneous execution of operations over vectors of data. This work presents the advances on the applicability of AVX2 on the development of prime field arithmetic, which is a building block for the construction of Elliptic Curve Cryptosystems. Having as a goal the efficient and secure implementation of prime field arithmetic, we show some advantages that vector instructions offer compared against 64-bit implementations. In order to validate the results of our research, we present a benchmark obtained on a Haswell processor.*

1. Research context.

Our research is focused on the efficient implementation of prime field arithmetic, we aim to use the most efficient techniques that can benefit from the capabilities of the recent micro-architectures. Implementing prime field operations not only involves the correctness of operations, but also efficient and secure processing. The first goal is achieved by speeding up operations, extracting parallelism over data and/or using a special instruction set. Nonetheless, in order to meet the security requirement, the implementation requires a detailed information flow analysis, also to avoid secret-dependent code branching and avoiding calculations that could reveal fragments of secret data. In this work, we accomplish both requirements, the first one through the use of AVX2 vector instructions and the second one through the development of constant time execution code.

2. The vector instruction set: AVX2.

Observing the trend of contemporary processors, most of them have replicated execution units to accomplish with out-of-order execution, thus exploiting the instruction level parallelism present on programs. Another interesting trend on the micro-architectures design is the use of SIMD (Single Instruction Multiple Data) processing, i.e. processors are provided of vector instructions that simultaneously compute an operation on every element of vector registers. Haswell micro-architecture is an example of this trending, it includes sixteen 256-bit registers (hereafter referred as `YMM` registers) and is the first one to support the AVX2 vector instruction set.

The AVX2 set includes instructions mostly oriented to perform integer arithmetic operations, variable-shift on registers and permutations of 64-bit words between registers. The release of AVX2 extends most of the integer arithmetic from 2 to 4 simultaneous operations per instruction. Instructions for integer arithmetic are so attractive for the implementation of prime field arithmetic, where usually the size of operands implies the use of multi-precision arithmetic, i.e. the size of operands is greater than the size of the native word machine (nowadays 64 bits).

3. Prime field arithmetic.

Prime fields are denoted as $\mathbb{F}_p$ where p is a prime number. Usually, the elements of $\mathbb{F}_p$ are represented by the integers in the set $\{0, 1, \ldots, p-1\}$. Addition (ADD), subtraction (SUB) and multiplication (MUL) of elements are performed modulo p. Modular multiplication is processed in two steps: first, the integer multiplication (iMUL) of both inputs is computed, and secondly a modular reduction (MOD) is performed. The special case of integer multiplication when both inputs are equal, it is known as integer squaring (iSQR).

We focus on the application of prime fields for the construction of Elliptic Curve Cryptography (ECC) schemes. ECC is well known to provide stronger security with shorter key lengths when compared to the RSA cryptosystem. Recently, new proposals for selecting parameters of elliptic curves and prime fields were published, such as [Bernstein 2006, Bos et al. 2014, Aranha et al. 2013]. These proposals claim that such new parameters will accelerate the execution performance of prime field operations. Table 1 shows the prime fields recently proposed and also the prime field currently used in standardized ECC by NIST[1] [Gallagher et al. 2009].

4. The radix-R representation.

Here is presented an efficient representation of prime field elements, called *radix*-64. In order to understand what *radix*-R is, first we will show two examples of commonly used representations:

1. The size of primes is always greater than 64 bits, which is the size of registers in commodity processors. We can use an array of 64-bit words to store elements of the prime field. This kind of approach is commonly used in multi-precision mathematical libraries and is also known as *radix-64* representation.
2. In [Bernstein 2006], author proposes the use of *radix-25.5*, for which an element $A \in \mathbb{F}_p$ is represented by the following polynomial: $A(x) = \sum_{i=0}^{k-1} a_i x^i$ where k is the number of floating point registers used to represent that element and each a_i is bounded according to the precision of floating point registers.

These representations can be generalized to *radix*-R representation, thus an element $A \in \mathbb{F}_p$ is represented by the following polynomial: $A(x) = \sum_{i=0}^{k-1} a_i x^i$ where $a_i \in [0, 2^R)$ are integer coefficients and $k = \left\lceil \frac{\lg(p)}{R} \right\rceil$ is the number of R-bit words used to represent A. Now, we will describe the algorithms used to compute prime field operations using *radix*-R representation:

- **Addition/Subtraction.** Given two elements A and B on radix-R representation we can compute $C = A \pm B$ as $c_i = a_i \pm b_i$ for $i \in [0, k)$. Notice that these operations are totally independent and admit a parallel processing.
- **Integer multiplication.** It computes an intermediate result $C_{i+j} \leftarrow \sum a_i b_j$ for $i, j \in [0, k)$, here k^2 word multiplications are processed. These operations have no carry dependencies between them.
- **Modular reduction**. When pseudo-Mersenne primes are used ($p = 2^m - c$), modular reduction only requires to process $C_i = C_i + cC_{i+k}$, for $i \in [0, k)$. Notice that for these primes modular reduction can be done faster than for the NIST's primes.

[1]NIST stands for National Institute of Standards and Technology.

5. Efficient implementation using AVX2.

As one can see, we can benefit from the parallelism presented on the operations . Now, we will present an efficient and secure implementation of prime field arithmetic in *radix*-R representation using AVX2 instructions. A similar work of this implementation is found in [Bernstein and Schwabe 2012], where NEON vector instructions were used to accelerate cryptographic primitives using an ARM architecture.

Since a `YMM` register stores four 64-bit words, our implementation uses $t = \lceil \frac{k}{4} \rceil$ `YMM` registers to store the integer coefficients of *radix*-64 representation. In order to compute modular addition, the AVX2 instruction set contain the `VPADDQ` (`VPSUBQ` for subtraction) instruction that computes four simultaneous 64-bit additions. However the last carry bit of each addition is lost. In order to overcome this issue, we restrict the R parameter to be $R < 64$, so each 64-bit operation has at least an extra available bit to store the carry bit produced by the addition operation. This restriction also applied to the case of integer multiplication. In order to compute $A \times B$, one has to add k intermediate products $a_i b_j$ for $i, j \in [0, k)$. To determine a bound for R we have:

$$\begin{aligned} k(2^R - 1)^2 &< 2^{64} \\ \log_2(k) + 2R &< 64 \\ \log_2(\log_2(p)) - \log_2(R) + 2R &< 64 \\ R - \tfrac{1}{2}\log_2(R) &< \tfrac{1}{2}(64 - \log_2(\log_2(p))). \end{aligned} \tag{1}$$

Then, the larger integer that holds (1) is $R = 30$, which nicely fits with the interface of `VPMULDQ` instruction. This instruction performs four simultaneous 32×32 bit multiplications. Finally, in the computation of the modular reduction, the terms cC_{i+k} are computed using shifts on vector registers instead of multiplications, and this can be done easily through the use of `VPSLLQ` and `VPSRLQ` instructions.

6. Preliminary results.

In the Table 1, we show the timings for the main operations on prime fields. The *radix-64* row refers to the implementation that uses native 64-bit instructions, such as a 64×64 bit multiplier (`MULX` instruction) and a 64-bit adder that computes addition with carry (`ADC` instruction). The results of our AVX2 implementation are shown in the *vec-radix-30* row.

We highlight that most of our timings using AVX2 instructions are competitive with the radix-64 implementation, for example, a modular multiplication (MUL) using the Curve25519's prime can be computed in 52 clock cycles on *radix-64*; while using the AVX2 implementation, it takes only 53 clock cycles, achieving almost the same performance. For the case of modular squaring (SQR), when is compared to *radix-64* implementation, our implementation is faster by 4 and 9 clock cycles for the Curve25519 and Curve1174 prime fields, respectively.

Modular addition and subtraction operations present almost the same performance 8-9 clock cycles. For vector implementation, we have that $R < 32$, this allows to compute more than one modular addition before a coefficient reduction be needed. The *coefficient reduction* is an operation that reduces each coefficient to the range $[0, 2^R)$ propagating the carries to the next significant coefficient.

			NISTp256		Curve25519		Curve1174	
			$\mathbb{F}_{2^{256}-2^{224}+2^{192}+2^{96}-1}$		$\mathbb{F}_{2^{255}-19}$		$\mathbb{F}_{2^{251}-9}$	
	iMUL	iSQR	ADD	MOD	ADD	MOD	ADD	MOD
radix-64	37	30	14	53	8	15	8	15
vec-radix-30	35	23	9	60	9	18	9	13

Table 1. Clock cycles measured to process each prime field operation on a Haswell processor Intel Core i7-4770.

The idea behind *radix-R* representation is to enable parallel computation that AVX2 vector instructions can take advantage. Our results show that the use of AVX2 is worthwhile on the implementation of prime field arithmetic. However, this representation also presents some side issues that results on additional operations, such as the coefficient reduction which takes around 28 clock cycles to be computed.

We keep investigating on optimization techniques for coefficient reduction and the application of the lazy reduction technique in order to minimize the impact of this modular operation. In order to compare our results against other implementations we will make a proof of concept on an elliptic curve cryptography protocol.

Acknowledgments: We would like to thank the anonymous reviewers for their helpful suggestions and comments. This research was supported in part by the Intel Labs University Research Office.

References

Aranha, D. F., Barreto, P. S. L. M., Pereira, G. C. C. F., and Ricardini, J. E. (2013). A note on high-security general-purpose elliptic curves. Cryptology ePrint Archive, Report 2013/647. `http://eprint.iacr.org/`.

Bernstein, D. and Schwabe, P. (2012). NEON Crypto. In Prouff, E. and Schaumont, P., editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 320–339. Springer Berlin Heidelberg.

Bernstein, D. J. (2006). Curve25519: New Diffie-Hellman Speed Records. In Yung, M., Dodis, Y., Kiayias, A., and Malkin, T., editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer.

Bos, J. W., Costello, C., Longa, P., and Naehrig, M. (2014). Selecting Elliptic Curves for Cryptography: An Efficiency and Security Analysis. Cryptology ePrint Archive, Report 2014/130. `http://eprint.iacr.org/`.

Gallagher, P., Foreword, D. D., and Director, C. F. (2009). FIPS PUB 186-3 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Digital Signature Standard (DSS).